Modeling Others using Oneself in Multi-Agent Reinforcement Learning

Roberta Raileanu [1] Emily Denton [1] Arthur Szlam [2] Rob Fergus [1,2]

Abstract

We consider the multi-agent reinforcement learning setting with imperfect information in which each agent is trying to maximize its own utility. The reward function depends on the hidden state (or goal) of both agents, so the agents must infer the other players' hidden goals from their observed behavior in order to solve the tasks. We propose a new approach for learning in these domains: Self Other-Modeling (SOM), in which an agent uses its own policy to predict the other agent's actions and update its belief of their hidden state in an online manner. We evaluate this approach on three different tasks and show that the agents are able to learn better policies using their estimate of the other players' hidden states, in both cooperative and adversarial settings.

1. Introduction

Reasoning about other agents' intentions and being able to predict their behavior is important in multi-agent systems, in which the agents might have a diverse, and sometimes competing, set of goals. This remains a challenging problem due to the inherent non-stationarity of such domains.

In this paper, we introduce a new approach for estimating the other agents' unknown goals from their behavior and using those estimates to choose actions. We demonstrate that in the proposed tasks, using an explicit model of the other player in the game leads to better performance than simply considering the other agent to be part of the environment.

We frame the problem as a (not-necessarily zero-sum) two-player stochastic game (Shapley, 1953), otherwise known as a two-player Markov game, in which the agents have full visibility of the environment, but no explicit knowledge about other agents' goals and there is no communication channel. The reward received by each agent at the end of an episode depends on the goals of both agents, so the optimal policy of each agent must take into account both of their goals.

Research in cognitive science suggests that humans maintain models of other people they interact with, which capture their goals, beliefs, or preferences (Gopnik & Wellman, 1992; Premack & Woodruff, 1978). In some cases, humans use their own mental process to simulate others' behavior by adopting their perspective (Gordon, 1986; Gallese & Goldman, 1998). This allows them to understand others' intentions or motives and act accordingly in social settings. Inspired by these studies, the key idea of our approach is that as a first approximation, to understand what the other player in the game is doing, an agent should ask itself "what would be my goal if I had acted as the other player had?". We instantiate this idea by parametrizing the agent's action and value functions with a (multi-layer recurrent) neural network that takes the state and a goal as an input. As the agent plays the game, it infers the other agent's unknown goal by directly optimizing over the goal (using its own action function) to maximize the likelihood of the other's actions.

2. Approach

Background: A Markov game for two agents is defined by a set of states $\mathcal{S}$ describing the possible configurations of all agents, a set of actions $\mathcal{A}_1, \mathcal{A}_2$ and a set of observations $\mathcal{O}_1, \mathcal{O}_2$ for each agents, and a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \to \mathcal{S}$ which gives the probability distribution on the next state as a function of current state and actions. Each agent i chooses actions by sampling from a stochastic policy $\pi_{\theta_i} : \mathcal{S} \times \mathcal{A}_i \to [0, 1]$. Each agent has a reward function which depends on agent's state and action: $r_i : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$. Each agent i tries to maximize its own total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$, where γ is a discount factor and T is the time horizon. In this work, we consider both cooperative, as well as adversarial settings.

We now describe Self Other-Modeling (SOM), a new approach for inferring the other agents' goals in an online fashion during an episode and using these estimates to choose actions. To decide an action and to estimate the value of a state, we use a neural network f that takes as input its own goal z_{self}, an estimate of the other player's goal $\tilde{z}_{other}$, and

[1]Department of Computer Science, New York University, New York City, New York, USA [2]Facebook AI Research, Facebook Inc., New York City, New York, USA. Correspondence to: Roberta Raileanu <raileanu@cs.nyu.edu>.

the observation state from its own perspective s_{self}, and outputs a probability distribution over actions π and a value estimate V, such that for each agent i playing the game we have:

$$\begin{bmatrix} \pi^i \\ V^i \end{bmatrix} = f^i(s^i_{self}, z^i_{self}, \tilde{z}^i_{other};\ \theta^i)\ .$$

Here θ^i are agent i's parameters for f, which has one soft-max output for the policy, one linear output for the value function, and all the non-output layers shared. The actions are sampled from the policy π. The observation state s^i_{self} explicitly contains the location of the acting agent (the one whose action is decided by f^i), as well as the location of the other agent.

Because an agent computes both its own actions and values, as well as estimates of the other agent's, each agent has two networks (omitting the agent index i for brevity):

$$f_{self}(s_{self}, z_{self}, \tilde{z}_{other};\ \theta_{self}) \qquad (1)$$

and

$$f_{other}(s_{other}, \tilde{z}_{other}, z_{self};\ \theta_{self})\ . \qquad (2)$$

The two networks are used in different ways: f_{self} is used for computing the agent's own actions and values, and operates in a feed-forward manner. The agent uses f_{other} to infer the other agent's goal via an optimization over $\tilde{z}_{other}$ given the other agent's observed actions.

We propose that each agent models the behavior of the other player using its own policy, so that the parameters of f_{other} are the same as the parameters of f_{self}. However, note that the two networks differ in their relative placement of the inputs z_{self} and $\tilde{z}_{other}$. Additionally, since the environment is fully observed, the observation state of the two agents differs only by the specification of the agent's identity on the map (i.e. each agent will be able to distinguish between its own location and the other's location). Hence, in acting mode, the network f_{self} will take as input s_{self} and in inference mode, the network f_{other} will take as input s_{other}.

At each step of the game, the agent needs to infer $\tilde{z}_{other}$ in order to input its estimate into (1) and choose its action. For this purpose, at each step, the agent observes the other taking an action and, at the next step, the agent uses the previously observed action of the other as supervision, in order to back-propagate through (2) and optimize over $\tilde{z}_{other}$. Figure 1 illustrates this technique.

The number of steps taken by the optimizer in this inference procedure is a hyperparameter that can be varied depending on the game. Hence, the estimate of the other agent's goal $\tilde{z}_{other}$ is updated multiple times at each step during the game. The parameters θ_{self} are updated at the end of each episode using Asynchronous Advantage Actor-Critic (A3C)

Algorithm 1 SOM training for one episode

1: **procedure** SELF OTHER-MODELING
2:     **for** k := 1, num_players **do**
3:         $\tilde{z}^k_{other} \leftarrow \frac{1}{ngoals}\mathbf{1}_{ngoals}$
4:     game.reset()
5:     **for** step := 1, episode_length **do**
6:         $i \leftarrow game.get\_acting\_agent()$
7:         $j \leftarrow game.get\_non\_acting\_agent()$
8:         $s^i_{self} \leftarrow game.get\_state()$
9:         $s^j_{other} \leftarrow game.get\_state()$
10:        $\tilde{z}^{OH,i}_{other} = one\_hot[argmax(\tilde{z}^i_{other})]$
11:        $\pi^i_{self}, V^i_{self} \leftarrow f^i_{self}(s^i_{self}, z^i_{self}, \tilde{z}^{OH,i}_{other}; \theta^i_{self})$
12:        $a^i_{self} \sim \pi^i_{self}$
13:        $game.action(a^i_{self})$
14:        **for** k : = 1, num_inference_steps **do**
15:            $\tilde{z}^{G,j}_{other} = gumbel\_softmax(\tilde{z}^j_{other})$
16:            $\tilde{\pi}^j_{other} \leftarrow f^j_{other}(s^j_{other}, \tilde{z}^{G,j}_{other}, z^i_{self}; \theta^j_{self})$
17:            $loss = cross\_entropy\_loss(\tilde{\pi}^j_{other}, a^i_{self})$
18:            $loss.backward()$
19:            $update(\tilde{z}^j_{other})$
20:     **for** k := 1, num_players **do**
21:        $policy.update(\theta^k_{self})$

(Mnih et al., 2016) with reward signal obtained by the self agent.

Algorithm 1 represents the pseudo-code for training SOM agents for one episode. Since the goals are discrete in all the tasks considered here, the agent's goal $\tilde{z}_{self}$ is encoded as a one-hot vector of dimension equal to the total number of possible goals in the game. The embedding of the other player's goal $\tilde{z}_{other}$ has the same dimension. In order to estimate the gradients going through $\tilde{z}_{other}$, which is a discrete variable and thus non-differentiable, we replace it with a differentiable sample from the Gumbel-Softmax distribution (Jang et al., 2016; Maddison et al., 2016), $\tilde{z}^G_{other}$. This reparametrization trick was shown to efficiently produce low-variance biased gradients. After optimizing $\tilde{z}_{other}$ at each step using this method, $\tilde{z}_{other}$ usually deviates from a one-hot vector. At the next step, f_{self} takes as input the one-hot vector $\tilde{z}^{OH}_{other}$ corresponding to the *argmax* of the previously updated $\tilde{z}_{other}$.

The agents' policies are parametrized by long short-term memory (LSTM) cells (Hochreiter & Schmidhuber, 1997) with two fully-connected linear layers, and exponential linear unit (ELU) (Clevert et al., 2015) activations. The weights of the networks are initialized with semi-orthogonal matrices, as described in (Saxe et al., 2013) and zero bias.

Due to the recurrence of f_{other}, special care must be taken when the number of inference steps is > 1. Under this
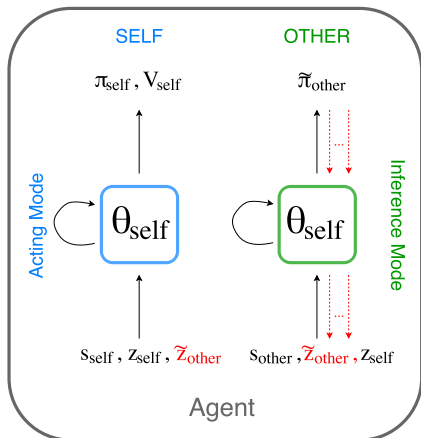
Figure 1. Our Self Other-Model (SOM) architecture for a given agent.

setting, at each step in the game, we save the recurrent state of f_{other} before the first forward pass in inference mode, and initialize the recurrent state to this value for every inference step. This procedure ensures f_{other} is unrolled the same number of steps during both acting and inference mode.

3. Related Work

Opponent modeling has been extensively studied in games of imperfect information. However, most previous approaches focus on developing models with domain-specific probabilistic priors or strategy parametrizations. In contrast, our work proposes a more general framework for opponent modeling. (Davidson, 1999) uses an MLP to predict opponent actions given a game history, but the agents cannot adapt to their opponents' behavior in an online manner. (Lockett et al., 2007) designs a neural network architecture to identify the opponent type by learning a mixture of weights over a given set of cardinal opponents. However, the game does not unfold within the reinforcement learning framework.

A large body of work in deep multi-agent RL focuses on partially visible, fully cooperative settings (Foerster et al., 2016a;b; Omidshafiei et al., 2017) and emergent communication (Lazaridou et al., 2016; Foerster et al., 2016a; Sukhbaatar et al., 2016; Das et al., 2017; Mordatch & Abbeel, 2017) Our setting is different since we do not allow any communication among the agents, so the players have to indirectly reason about their opponents' intentions from their observed behavior. In contrast, (Leibo et al., 2017) considers semi-cooperative multi-agent environments in which the agents develop cooperative and competitive strategies depending on the task type and reward structure. Similarly, (Lowe et al., 2017) proposes a centralized actor-critic architecture for efficient training in settings with such

mixed strategies. (Lerer & Peysakhovich, 2017) design RL agents that are able to maintain cooperation in complex social dilemmas by generalizing a well-known game theoretic strategy called tit-for-tat (Axelrod, 2006) to multi-agent Markov games. Recent work in cognitive science attempts to understand human decision-making by using a hierarchical model of social agency that infers the intentions of other human agents in order to decide whether to play a cooperative or competitive strategy (Kleiman-Weiner et al., 2016). However, none of these papers design algorithms that explicitly model other artificial agents in the environment or estimate their intentions, with the purpose of improve their decision making.

The field of inverse reinforcement learning (IRL) (Russell, 1998; Ng et al., 2000; Abbeel & Ng, 2004), is also related to the problem considered here. IRL's aim is to infer the reward function of an agent by observing its behavior, which is assumed to be nearly optimal. In contrast, our approach uses the observed actions of the other player to directly infer its goal in an online manner, which is then used by the agent when acting in the environment. This avoids the need for collecting offline samples of the other's (state, action) pairs in order to estimate its reward function and then use this to learn a separate policy that maximizes that utility. The more recent papers by (Hadfield-Menell et al., 2016; 2017) are also concerned with the problem of inferring others' intentions, but their focus is on human-robot interaction and value alignment. Motivated by similar goals, (Chandrasekaran et al., 2017) consider the problem of building a theory of AI's mind, in order to improve human-AI interaction and the interpretability of AI systems. For this purpose, they show that people can be trained to predict the responses of a Visual Question Answering model, using a small number of examples.

The closest work to ours is (Foerster et al., 2017) and (He et al., 2016). (Foerster et al., 2017) designs RL agents that take into account the learning of other agents in the environment when updating their own policies. This enables the agents to discover self-interested yet collaborative strategies such as tit-for-that in the iterated prisoner's dilemma. While our work does not explicitly attempt to shape the learning of other agents, it has the advantage that the agents can update their beliefs during an episode and change their strategies in an online manner to gain more reward. Our setting is also different in that it considers that each agent has some hidden information needed by their the other player in order to maximize its return.

Our work is very much in line with (He et al., 2016), where the authors build a general framework for modeling other agents in the reinforcement learning setting. (He et al., 2016) proposes a model that jointly learns a policy and the behavior of opponents by encoding observations of the

opponent into a DQN. Their Mixture of Experts architecture is able to discover different opponent strategy patterns in two purely adversarial tasks. One difference between our work and (He et al., 2016)'s is that we do not aim to infer other agents' strategies, but rather focus on explicitly estimating their goals in the environment. Moreover, rather than using a hand designed featurization of the other agent's actions, in this work, the agent learns its model of the other end-to-end, based on its own model. Another difference is that in this work, the agent runs an optimization to infer the other agent's hidden state, instead of inferring the other agent's hidden state via a feed-forward network. In the experiments below, we show that SOM outperforms an adaptation of the method of (He et al., 2016) to our setting.

4. Experiments

In this section, we evaluate our model SOM on three tasks:

- The coin game, in Section 4.2, which is a fully cooperative task where the agents' roles are symmetric.

- The recipe game, in Section 4.3, which is adversarial, but with symmetric roles.

- The door game, in Section 4.4, which is fully cooperative but has asymmetric roles for the two players.

We compare SOM to three other baselines and to a model that has access to the ground truth of the other agent's goal. All the tasks considered are created in the Mazebase gridworld environment (Sukhbaatar et al., 2015).

4.1. Baselines

TRUE-OTHER-GOAL (TOG): We provide an upper bound on the performance of our model given by a policy network which takes the other agent's **true** goal as input, z_{other}, as well as the state features s_{self} and its own goal z_{self}. Since this model has direct access to the true goal of the other agent, it does not need a separate network to model the behavior of the other agent. The architecture of TOG is the same as the one of SOM's policy network, f_{self}.

NO-OTHER-MODEL (NOM): The first baseline we use only takes as inputs the observation state s_{self} and its own goal z_{self}. NOM has the same architecture as the one used for SOM's policy network, f_{self}. This baseline has no explicit model of the other agent or estimate of its goal.

INTEGRATED-POLICY-PREDICTOR (IPP): Starting with the architecture and inputs of NOM, we construct a stronger baseline, IPP, which has an additional final linear layer that outputs a probability distribution over the next action of the other agent. Besides the A3C loss used to train the policy of this network, we also add a cross-entropy loss to train the prediction of the other agent's action, using observations of its behavior.

SEPARATE-POLICY-PREDICTOR (SPP): He et al. (2016) propose an opponent modeling framework based on DQN. In their approach, a neural network (separate from the learned Q-network) is trained to predict the opponent's actions, given hand crafted state information specific to the opponent. An intermediate hidden representation from this network is given as input to the the Q-network.

We adapt the model of He et al. (2016) to our setting. In particular, we use A3C instead of DQN and we do not use the task-specific features used to represent the hidden state of the opponent.

The resulting model, SPP, consists of two *separate* networks, a policy network for deciding the agent's actions, and an opponent network for predicting the other's actions. The opponent network takes as input the state of the world s and its own goal z_{self}, and outputs a probability distribution for the action taken by the other agent at the next step, as well as its hidden state (given by the network's recurrence). As in IPP, we train the opponent policy predictor with a cross-entropy loss using the true actions of the other agent. At each step, the hidden state output by this network is taken as input by the agent's policy network, along with the observation state and its own goal. Both the policy network and the opponent policy predictor are LSTMs with the same architecture as SOM.

In contrast to SOM, SPP does not explicitly infer the other agent's goal. Rather, it builds an implicit model of the opponent by predicting the agent's actions at each time step. In SOM, an inferred goal is given as additional input to the policy network. The analog of the inferred goal in SPP is the hidden representation obtained from the opponent policy predictor which is given as an additional input to the policy network.

Training Details. In all our experiments, we train the agents' policies using A3C (Mnih et al., 2016) with an entropy coefficient of 0.01, a value loss coefficient of 0.5, and a discount factor of 0.99. The parameters of the agents' policies are optimized using Adam (Kingma & Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$, and weight decay 0. SGD with a learning rate of 0.1 was used for inferring the other agent's goal, $\tilde{z}_{other}$.

The hidden layer dimension of the policy network was 64 for the Coin and Recipe Games and 128 for the Door Game. We use a learning rate of 1×10^{-4} for all the games and models.

The observation state s is represented by few-hot vectors indicating the locations of all the objects in the environment, as well as the locations of the self and the other. The dimen-

sion of this input state is $1 \times nfeatures$, where the number of features is 384, 192, and 900 for the Coin, Recipe, and Door games, respectively.

For each experiment, we trained the models using 5 different random seeds. All the results shown are for 10 optimization updates of $\tilde{z}_{other}$ at each step in the game, unless mentioned otherwise.

4.2. Coin Game.

First, we evaluate the model on a fully cooperative task, in which the agents can gain more reward when using both of their goals rather than only their own goal. So it is in the best interest of each agent to estimate the other player's goal and use that information when taking actions. The game, shown in the left diagram of Figure 4, takes place on a 8×8 grid containing 12 coins of 3 different colors (4 coins of each color). At the beginning of each episode, the agents are randomly assigned one of the three colors. The action space consists of: go up, down, left, right, or pass. Once an agent steps on a coin, that coin disappears from the grid. The game ends after 20 steps (i.e. each agent takes 10 steps). The reward received by both agents at the end of the game is given by the formula below:

$$R = (n_{C_{self}}^{self} + n_{C_{self}}^{other})^2 + (n_{C_{other}}^{self} + n_{C_{other}}^{other})^2$$
$$- (n_{C_{neither}}^{self} + n_{C_{neither}}^{other})^2,$$

where $n_{C_{self}}^{other}$ is the number of coins of the self's goal-color, which were collected by the other agents, and $n_{C_{neither}}^{self}$ is the number of coins corresponding to neither of the agents' goals, collected by the self. For the example in Figure 4, agent 1 has C_{self} = orange and C_{other} = cyan, while agent 2's C_{self} is cyan and C_{other} is orange. $C_{neither}$ is red for both agents.

The role of the penalty for collecting coins that do not correspond to any of the agents' goals is to avoid convergence to a brute force policy in which the agents can gain a non-negligible amount of reward by collecting all the coins in their proximity, without any regard to their color.

To maximize its return, each agent needs to collect coins of its own or its collaborator's color, but not those of the remaining color. Thus, when both agents are able to infer their collaborators' goals with high accuracy and as early as possible in the game, they can use that information to maximize their shared utility.

Figure 3 shows the mean and standard deviation of the reward across 5 runs with different random seeds obtained by SOM. Our model clearly outperforms all the baselines on this task. We also show the empirical upper bound on the reward using the model which takes as input the true color assigned to the other agent.

Figure 2 analyzes the strategies of the different models by looking at the proportion of coins of each type collected by the agents. The optimal strategy is for each agent to maximize $n_{C_{self}}^{self} + n_{C_{other}}^{self}$ and $n_{C_{neither}}^{self} = 0$. Due to the randomness in the initial conditions (in particular, the locations of coins in the environment), this amounts to each agent collecting an equal number of coins of its own color and coins of the other's color on average, across a large number of episodes (i.e. $\bar{n}_{C_{self}}^{self} = \bar{n}_{C_{other}}^{self}$).

Indeed, this is the strategy learned by the model with perfect information of the other agent's goal (TOG). SOM also learns to collect significantly more Other than Neither coins (although not as many as Self coins), indicating its ability to distinguish between the two types, at least during some of the episodes. This means that SOM can accurately infer the other agent's goal early enough during the episode and use that information to collect more Other Coins, thus gaining more reward than if it were only using its own goal to direct its actions.

In contrast, the agents trained with the three baseline models collect significantly more Self coins, and as many Other as Neither coins on average. This shows that they learn to use their own goal for gaining reward, but they are unable to use the hidden goal of the other agent for further increasing their reward. Even if IPP and SPP are able to predict the actions of the other player with an accuracy of about 50%, they do not learn to distinguish between the coins that would increase (Other) and those that would decrease (Neither) their reward. This shows the weaknesses of using an implicit model of the other agent to maximize reward on certain tasks.

4.3. Recipe Game.

Agents in adversarial scenarios can also benefit from having a model of their opponents, which would enable them to exploit the weaknesses of certain players. With this motivation in mind, we evaluate our model on a game in which the agents have to craft certain compositional recipes, each containing multiple items found in the environment. The agents are given as input the names of their goal-recipes, without the corresponding components needed to make it. The resources in the environment are scarce, so only one of the agents can craft its recipe within one episode.

As illustrated in Figure 4 (center), there are 4 types of items: {sun, star, moon, lightning} and 4 recipes: {sun, sun, star}; {star, star, moon}; {moon, moon, lightning}; {lightning, lightning, sun}. The game is played in a 4×6 grid, which contains 8 items in total, 2 of each type.

At the beginning of each episode, we randomly assign a recipe to one of the agents, and then we randomly pick a recipe for the other agent so that it has overlapping items with the recipe of the first agent. This ensures that the agents
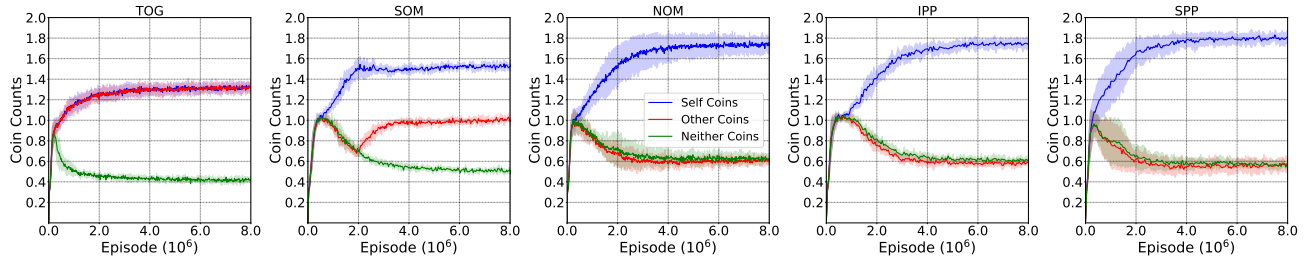
Figure 2. **Coin Strategy:** Average number of collected coins per episode corresponding to the color of the Self (blue), Other (red), or Neither (green) by the agents using TOG (left), SOM (center-left), NOM (center), IPP (center-right), and SPP (right). The optimal strategy is to pick up as many Self as Other coins on average, across a number of episodes, and no Neither coins. Being able to collect more Other than Neither coins indicates that the agent is able to accurately infer the other agent's color early enough during some of the episodes and uses this information to collect more Other, instead of Neither coins, which increases its reward. The TOG model learns to collect just as many Self as Other coins, while all the baselines only learn to collect more Self coins, but cannot distinguish between the Other and Neither coins. SOM learns to collect significantly more Other coins than Neither. This shows that SOM converges to a closer-to-optimal strategy using its guess of the other's goal.
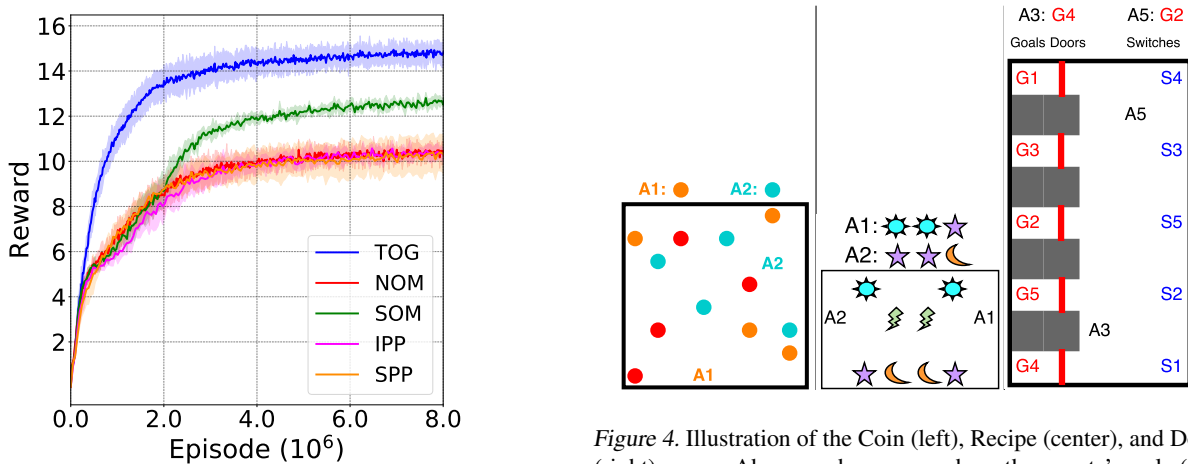

Figure 3. **Coin Performance:** Average reward obtained on the Coin game by SOM (green), TOG (blue), NOM (red), IPP (magenta), and SPP (orange). SOM performs better than all the baselines.

Figure 4. Illustration of the Coin (left), Recipe (center), and Door (right) games Above each ones we show the agents' goals (not visible to one another).

are competing for resources within each episode. At the end of the episode, each agent receives a reward of +1 for crafting its own recipe and a penalty of -0.1 for each item it picked up not needed for making its recipe.

We designed the layout of the grid so that neither agent has an initial advantage by being closer to the scarce resource. At the beginning of each episode, one of the agents starts on the left-most column of the grid, while the other one starts on the right-most column, at the same y-coordinate. Their initial y-coordinate as well as which agent starts on the left/right is randomized. Similarly, one item of each of the 4 different types is placed at random in the grid formed

by the second and third columns of the maze, from left to right. The rest of the items are placed in the forth and fifth columns, so that the symmetry with respect to the vertical axis is preserved (i.e. items of the same type are placed at the same y-coordinate, and symmetric x-coordinates).

Agents have six actions to choose from: pass, go up, down, left, right, or pick (for picking an item, which then disappears from the grid). The first agent to take an action is randomized. The game ends after 50 steps.

We pretrain all the baselines on a version of the game which does not have overlapping recipes, in order to ensure that all the models learn to pick up the corresponding items, given a recipe as goal. All of the models learn to craft their assigned recipes $\sim 90\%$ of the time on this simpler task. Then, we continue training the models on the adversarial task in which their recipes overlap in each episode. SOM is
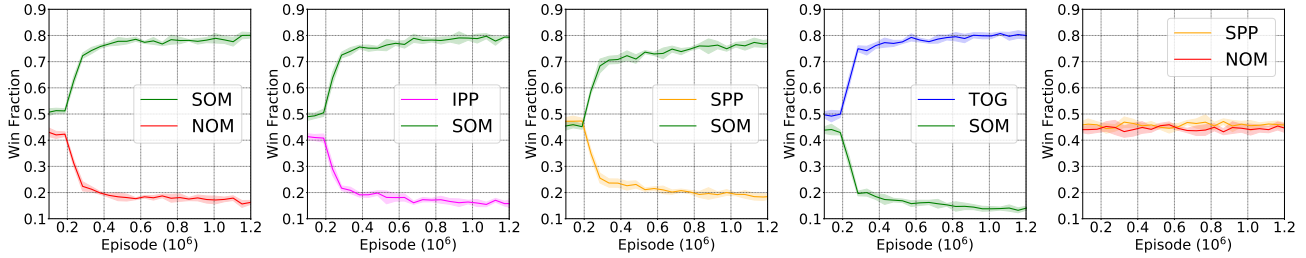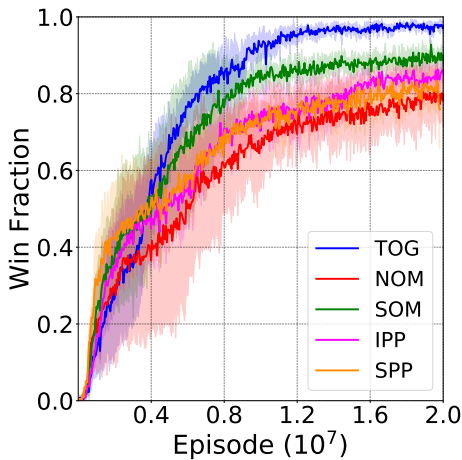
Figure 5. **Recipe Performance:** Average fraction of success in the Recipe game by SOM-NOM (left), SOM-IPP (center-left), SOM-SPP (center-center), SOM-TOG (center-right), NOM-AcrPredSep (right). The plots show the performance of SOM with 5 optimization updates of $\tilde{z}_{other}$ at each step in the game.

initialized with a pretrained NOM network.

Figure 5 shows the winning fraction for different pairs played against each other in the Recipe game. For the first 100k episodes, the models are not being trained. We can see that SOM significantly outperfroms NOM, IPP, and SPP, winning $\sim 75 - 80\%$ of the time, while the baselines can only win $\sim 15 - 20\%$ of the games. SPP ties against NOM, and TOG outperforms SOM by a large margin. We also played the same types of agents against each other and they all win $\sim 40 - 50\%$ of the games.

4.4. Door Game.

In this section, we show that on a collaborative task with asymmetric roles and multiple possible partners, the agents can learn to figure out what role they should be playing in each game based on their partners actions.

Figure 6. **Door Performance:** Average fraction of success on the Door game by SOM (green), TOG (blue), NOM (red), IPP (magenta), and SPP (orange). On average, SOM performs better than all the baselines.

In the Door game, two agents are located in a 5×9 grid,

with 5 goals behind 5 doors on the left wall, and 5 switches on the right wall of the grid. The game starts with the two players in random squares on the grid, except for the ones occupied by the goals, doors, or switches, we illustrated in Figure 4. Agents can take any of the five actions: go up, down, left, right or pass. An action is invalid if it moves the player outside of the border or to a square occupied by a block or closed door. Both agents receive +3 reward when either one of them steps on its goal and they are penalized -0.1 for each step they take. The game ends when one of them gets to its goal or after 22 steps. All the goals are behind doors which are open only as long as one of the agents sits on the corresponding switch for that door.

At the beginning of an episode, each of the two players is randomly selected from a pool of 5 agents and receives as input a random number from 1 to 5 corresponding to its goal. Each of the 5 agents has its own policy which gets updated at the end of each episode they play. Note that the agents' identities are not visible (i.e. there is no indication in the state features that specifies the id's of the agents playing during a given episode). This restriction is important in order to ensure that the agents cannot gain advantage by specializing into the two roles needed to win (i.e. goal-goer and switch-puller) and identifying the specialization of the other player by simply observing its unique id.

The agents need to cooperate in order to receive reward. In contrast to our previous tasks, the two players must take different roles. In fact, the player who sits on the switch should ignore its own goal and instead infer the other's goal, while the player who goes to its goal does not need to infer the other's goal, but only use its own. In order to sit on the correct switch, an agent has to infer the other player's goal from their observed actions. The only way in which an agent can use its own policy to model the other player is if each agent learns to play both roles of the game, i.e. go to its own goal and also open its collaborator's door by sitting on the corresponding switch. Indeed, we see that the agents learn to play both roles and they are able to use their own policies to infer the other player's goals when needed.

Fig 6 shows the mean and standard deviation of the winning fraction obtained by one of the agents on the Door game. While our model is still able to outperform the three baselines, the gap between the performance of our model and that of IPP or SPP (an approximate version of (He et al., 2016)) is smaller than in the previous tasks. However, this is a more difficult task for our model since it needs the agents to learn performing both roles before effectively use its own policy to infer the other agent's goal. Nevertheless, we see that SOM training allows the agents to play both roles in an asymmetric cooperative game, and to infer the goal and role of the other player.

4.5. Analyzing the goal inference

In this section we further analyze the ability of the SOM models to infer other's intended goals.
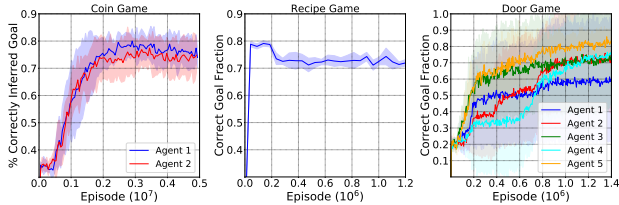
Figure 7. **Inference Accuracy during Training:** The mean fraction of episodes in which the agent correctly infers the other's goal for the Coin (left), Recipe (center), and Door (right) games, as a function of training epoch. The estimate of the other's goal is considered correct if it remains accurate during all the following steps in the game.
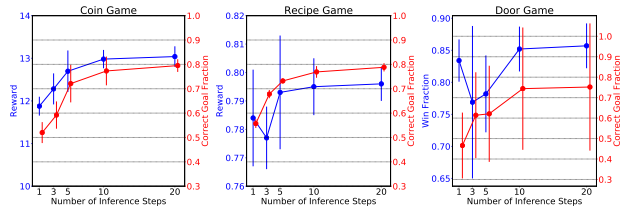
Figure 7 shows the fraction of episodes in which the goal of the other agent is correctly inferred. We consider that the goal is correctly inferred only when the estimate of the other's goal remains accurate until the end of the game, so that we avoid counting the episodes in which the agent might infer the correct goal by chance at some intermediate step in the game. In all the games, the SOM agent learns to infer the other player's goal with a mean accuracy ranging from $\sim 60 - 80\%$. Comparing the second plot in Figure 2 with the left plot in Figure 7, one can observe that the SOM agent starts distinguishing Other from Neither coins after approximately 2M training epochs, which coincides with the time when the mean accuracy of the inferred goal converges to $\sim 75\%$. The Door Game (right) presents higher variance since the agents learn to use and infer the other's goal at different stages during training.

Figure 8 shows the cumulative distribution of the step at which the goal of the other player is correctly inferred (and remains the same until the end of the game). The cumulative distribution is computed over the episodes in which the goal is correctly inferred before the end of the game. In the
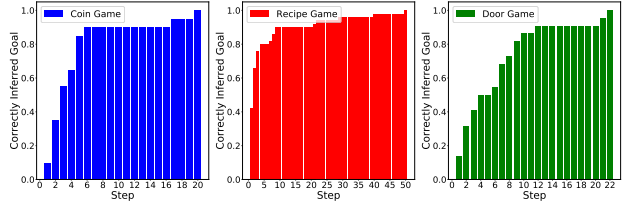
Figure 8. **Inference Step Distribution:** Cumulative distribution of the step t_{inf} at which the goal of the other player is correctly inferred (i.e. $\tilde{z}_{other}^t = z_{other}, \forall t \geq t_{inf}$) for the Coin (left), Recipe (center) and Door (right) games. We define this step so that $\tilde{z}_{other} = z_{other}$ for all the remaining steps in the game. The distribution is computed over the subset of runs in which the goal is correctly inferred before the end of the game ($\sim 70 - 80\%$ of all runs). A total of 1000 runs with trained SOM models were used to compute this distribution.

Coin (blue) and Recipe (red) games, 80% of the times the agent correctly infers the goal of the other, it does so in the first five steps. The distribution for the Door (green) game indicates that the agent needs more steps on average to correctly infer the goal. This explains in part why the SOM agent only slightly outperforms the SPP baseline. If the agent does not infer the other's goal early enough in the episode, it cannot efficiently use it to maximize its reward.

Figure 9. **Performance Variation with Number of Inference Steps:** Average reward (blue) and average fraction of episodes in which the goal of the other agent is correctly inferred (red) obtained by the SOM agent as a function of the number of inference steps used for estimating the other's goal for the Coin (left), Recipe (center), and Door (right) games. The error bars represent 1 standard deviation.

Figure 9 shows how the performance of the agent varies with the number of optimization updates performed on $\tilde{z}_{other}$ at each step in the game. As expected, the agent's reward (blue) generally increases with the number of inference steps, as does the fraction of episodes in which the goal is correctly inferred. One should note that increasing the number of inference steps from 10 to 20 only translates into less than 0.45% performance gain, while increasing it from 1 to 5 translates into a performance gain of 6.9% on the Coin game, suggesting that there is a certain threshold

above which increasing the number of inference steps will not significantly improve performance.

5. Discussion

In this paper, we introduced a new approach for inferring other agents' hidden states from their behavior and using those estimates to choose actions. We demonstrated that the agents are able to estimate the other players' hidden goals in both cooperative and competitive settings, which enables them to converge to better policies and gain higher rewards. In the proposed tasks, using an explicit model of the other player led to better performance than simply considering the other agent to be part of the environment. One limitation of SOM is that it requires a longer training time than the other baselines, since we back-propagate through the network at each step. However, their online nature is essential in adapting to the behavior of other agents in the environment.

Some of the main advantages of our method are its simplicity and flexibility. This method does not require any extra parameters to model the other agents in the environment, can be trained with any reinforcement learning algorithm, and can be easily integrated with any policy parametrization or network architecture. The SOM concept can be adapted to settings with more than two players, since the agent can use its own policy to model the behavior of any number of agents and infer their goals. Moreover, it can be easily generalized to many different environments and tasks.

We plan to extend this work by evaluating the models on more complex environments with more than two players, mixed strategies, a more diverse set of agent types (e.g. agents with different action spaces, reward functions, roles or strategies), and to model deviations from the assumption that the other player is just like the self.

Other important avenues for future research are to design models that can adapt to non-stationary strategies of others in the environment, handle tasks with hierarchical goals, and perform well when playing with new agents at test time.

Finally, many research areas could benefit from having a model of other agents that allows reasoning about their intentions and predicting their behavior. Such models might be useful in human-robot or teacher-student interactions (Dragan et al., 2013; Fisac et al., 2017), as well as for value alignment problems (Hadfield-Menell et al., 2016). Additionally, these methods could be useful for model-based reinforcement learning in multi-agent settings, since the accuracy of the forward model strongly depends on the ability of predicting others' behavior.

References

Abbeel, Pieter and Ng, Andrew Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Axelrod, Robert. Agent-based modeling as a bridge between disciplines. *Handbook of computational economics*, 2: 1565–1584, 2006.

Chandrasekaran, Arjun, Yadav, Deshraj, Chattopadhyay, Prithvijit, Prabhu, Viraj, and Parikh, Devi. It takes two to tango: Towards theory of ai's mind. *arXiv preprint arXiv:1704.00717*, 2017.

Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Das, Abhishek, Kottur, Satwik, Moura, José MF, Lee, Stefan, and Batra, Dhruv. Learning cooperative visual dialog agents with deep reinforcement learning. *arXiv preprint arXiv:1703.06585*, 2017.

Davidson, Aaron. Using artificial neural networks to model opponents in texas holdem. *Unpublished manuscript*, 1999.

Dragan, Anca D, Lee, Kenton CT, and Srinivasa, Siddhartha S. Legibility and predictability of robot motion. In *Human-Robot Interaction (HRI), 2013 8th ACM/IEEE International Conference on*, pp. 301–308. IEEE, 2013.

Fisac, Jaime F, Gates, Monica A, Hamrick, Jessica B, Liu, Chang, Hadfield-Menell, Dylan, Palaniappan, Malayandi, Malik, Dhruv, Sastry, S Shankar, Griffiths, Thomas L, and Dragan, Anca D. Pragmatic-pedagogic value alignment. *arXiv preprint arXiv:1707.06354*, 2017.

Foerster, Jakob, Assael, Yannis M, de Freitas, Nando, and Whiteson, Shimon. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016a.

Foerster, Jakob N, Assael, Yannis M, de Freitas, Nando, and Whiteson, Shimon. Learning to communicate to solve riddles with deep distributed recurrent q-networks. *arXiv preprint arXiv:1602.02672*, 2016b.

Foerster, Jakob N, Chen, Richard Y, Al-Shedivat, Maruan, Whiteson, Shimon, Abbeel, Pieter, and Mordatch, Igor. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

Gallese, Vittorio and Goldman, Alvin. Mirror neurons and the simulation theory of mind-reading. *Trends in cognitive sciences*, 2(12):493–501, 1998.

Gopnik, Alison and Wellman, Henry M. Why the child's theory of mind really is a theory. *Mind & Language*, 7 (1-2):145–171, 1992.

Gordon, Robert M. Folk psychology as simulation. *Mind & Language*, 1(2):158–171, 1986.

Hadfield-Menell, Dylan, Russell, Stuart J, Abbeel, Pieter, and Dragan, Anca. Cooperative inverse reinforcement learning. In *Advances in neural information processing systems*, pp. 3909–3917, 2016.

Hadfield-Menell, Dylan, Milli, Smitha, Abbeel, Pieter, Russell, Stuart J, and Dragan, Anca. Inverse reward design. In *Advances in Neural Information Processing Systems*, pp. 6768–6777, 2017.

He, He, Boyd-Graber, Jordan, Kwok, Kevin, and Daumé III, Hal. Opponent modeling in deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1804–1813, 2016.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kleiman-Weiner, Max, Ho, Mark K, Austerweil, Joseph L, Littman, Michael L, and Tenenbaum, Joshua B. Coordinate to cooperate or compete: abstract goals and joint intentions in social interaction. In *COGSCI*, 2016.

Lazaridou, Angeliki, Peysakhovich, Alexander, and Baroni, Marco. Multi-agent cooperation and the emergence of (natural) language. *arXiv preprint arXiv:1612.07182*, 2016.

Leibo, Joel Z, Zambaldi, Vinicius, Lanctot, Marc, Marecki, Janusz, and Graepel, Thore. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, pp. 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

Lerer, Adam and Peysakhovich, Alexander. Maintaining cooperation in complex social dilemmas using deep reinforcement learning. *arXiv preprint arXiv:1707.01068*, 2017.

Lockett, Alan J, Chen, Charles L, and Miikkulainen, Risto. Evolving explicit opponent models in game playing. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 2106–2113. ACM, 2007.

Lowe, Ryan, Wu, Yi, Tamar, Aviv, Harb, Jean, Abbeel, Pieter, and Mordatch, Igor. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

Maddison, Chris J, Mnih, Andriy, and Teh, Yee Whye. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Mordatch, Igor and Abbeel, Pieter. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

Ng, Andrew Y, Russell, Stuart J, et al. Algorithms for inverse reinforcement learning. In *Icml*, pp. 663–670, 2000.

Omidshafiei, Shayegan, Pazis, Jason, Amato, Christopher, How, Jonathan P, and Vian, John. Deep decentralized multi-task multi-agent rl under partial observability. *arXiv preprint arXiv:1703.06182*, 2017.

Premack, David and Woodruff, Guy. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1 (4):515–526, 1978.

Russell, Stuart. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pp. 101–103. ACM, 1998.

Saxe, Andrew M, McClelland, James L, and Ganguli, Surya. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Shapley, L. S. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. ISSN 0027-8424. doi: 10.1073/pnas.39.10.1095. URL `http://www.pnas.org/content/39/10/1095`.

Sukhbaatar, Sainbayar, Szlam, Arthur, Synnaeve, Gabriel, Chintala, Soumith, and Fergus, Rob. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.

Sukhbaatar, Sainbayar, Fergus, Rob, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pp. 2244–2252, 2016.